Transitory master key transport layer security for WSNs

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 27, 2019, accepted January 3, 2020, date of publication January 23, 2020, date of current version January 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2969050

Transitory Master Key Transport Layer Security for WSNs

MATTIA GRIOTTI[1], FILIPPO GANDINO[2], (Member, IEEE), AND MAURIZIO REBAUDENGO[2], (Senior Member, IEEE)

[1]TecSA s.r.l., 10067 Vigone, Italy
[2]Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy

Corresponding author: Filippo Gandino (filippo.gandino@polito.it)

ABSTRACT Security approaches in Wireless Sensor Networks (WSNs) are normally based on symmetric cryptography. Instead of symmetric encryption, some alternative approaches have been developed by using public-key cryptography. However, the higher computational cost represents a hard limitation to their use. In this paper, a new key management protocol is proposed. A transitory symmetric key is used to authenticate nodes in the network during the key establishment. However, pairwise keys are established using asymmetric cryptography. A theoretical analysis shows that the computational effort required by the public key cryptosystem is greatly reduced, while the security of the network is increased with respect to state-of-the-art schemes based on a transitory master key. Moreover, an experimental analysis demonstrates that this proposed approach can reduce the time spent for key establishment by about 35%.

INDEX TERMS Wireless sensor networks, asymmetric cryptography, key management, Internet of Things.

I. INTRODUCTION

Wireless sensor networks (WSNs) [1] are a well-established pervasive technology that represents an ideal sensing component in the Internet of things (IoT) [2]. They are composed of low cost and low power devices, called sensor nodes, which sense the environment, process the collected data and exchange information through a wireless connection. They are applied in numerous fields, like industry 4.0 [3], smart city [4] and air quality monitoring [5].

WSNs share fundamental characteristics with embedded systems, like low power devices with low cost hardware and special purpose applications. According to these characteristics and to their network system, ad-hoc solutions must be implemented to solve ordinary issues like power consumption [6], channel allocation [7] and reliability [8]. In particular, WSNs are affected by security threads (e.g., eavesdropping [9] and hardware tampering [10]).

Symmetric cryptography in WSNs is normally preferred to public cryptography, since it requires a lower computational effort. However, with symmetric cryptography two nodes can communicate only if they share a common secret, i.e. a *key*. Various *key management* approaches have been developed in order to establish and distribute keys in a WSN [11]. The *Plain global key* (PGK) represents the basic security solution. A single global key is used to encrypt all the communications. This scheme has very low memory overheads but it also provides a low security level because all nodes store the common secret. Therefore, if an adversary compromises a node the entire network is compromised. *Full pairwise key* (FPWK) is another basic approach. Also this scheme is based on key predistribution: before the deployment the keys are stored in the memories of the nodes. Additional computation or data exchange are not required. In this case, each possible link in the network has its own secret key. Therefore, each node stores a key per node in the network. The required memory is proportional to the size of the network, so FPWK can be only applied to small networks. However, if an adversary compromises a node he/she cannot use the achieved information to eavesdrop on the communications among the other nodes.

Within the techniques based on symmetric encryption, *transitory master key* represents a well-known solution for static networks. At deployment each node stores a global secret that is used to generate pairwise keys. However, after a small period of time each node deletes the global secret. In this way, if a node is compromised after the deletion, the rest of the network is safe. LEAP+ [12] is the main example of schemes based on this technique.

The communication security within WSNs is based on symmetric cryptography. Moreover, public key cryptography is also considered too expensive to protect the

The associate editor coordinating the review of this manuscript and approving it for publication was Gerhard P. Hancke.

key establishment, since the limited computational and power resources of the nodes clash with the involved computational overheads. However, public key could simplify the key management. An example of public key management scheme is the simplified versions of TLS (*Transport layer security*) [13], which is the protocol used on the Internet to establish secure connections. In this scheme, public key digital certificates are used for authentication and then the key establishment is done with key agreement functions (e.g. Diffie-Hellman).

This paper presents *Transitory master key TLS* (TMKTLS), an hybrid protocol based on both symmetric and asymmetric cryptography. Its main goal is to reduce the computational requirements for the application of public cryptography in WSNs. In TMKTLS, a temporary master key, shared by all nodes, is employed to authenticate the public keys of the nodes. After an initial time slot, the transitory master key is deleted and pairwise keys among the nodes are generated using asymmetric cryptography. This scheme authenticates the public keys with a message authentication code instead of a digital certificate; the former operation takes negligible time compared to the latter, so the overall time required for key establishment is greatly reduced. If an adversary compromises the transitory master key, he/she can only add fake nodes to the network, while data secrecy is always preserved by public cryptography. Moreover, these malicious nodes can be detected by a malicious node detection routine. This kind of technique has been also used in [14]. Differently from that approach, TMKTLS is compliant with node adding and mobile nodes, even if it has better performance with static networks.

A theoretical analysis shows that the computational effort required by TMKTLS is greatly lower than standard public key cryptosystems. The schemes based on transitory master key have common security limitations related to the possibility that the transitory master key is compromised. However, TMKTLS provides a good protection even if the transitory master key is compromised, since public cryptography still protects the links from eavesdropping and the malicious node detection routine allows to identify fake nodes. Moreover, an experimental analysis on real nodes validates the theoretical analysis and demonstrates that this new approach can decrease the time spent for key establishment up to a third.

The rest of the paper is organized as follows. In Sect. II related works are described. Sect. III presents the proposed key management scheme. In Sect. IV, the proposed approach is theoretically analyzed and compared with the state-of-the-art approaches, while in Sect. V, an experimental analysis validates the scheme. Finally, conclusions are drawn in Sect. VI.

II. RELATED WORKS

Many key management schemes based on different approaches have been presented in literature [15], [16]. In the following, the most important ones are described.

A. GLOBAL MASTER KEY

Schemes based on the Global Master Key approach use a unique master key that is shared by all the nodes and is used to protect all the communications or to provide security during the pairwise key establishment. The main approach in this category is *Symmetric-key Key Establishment* (SKKE) [17], which is the key management scheme used by ZigBee. In SKKE every node is preloaded with the master key. To generate a pairwise key between node A and node B, node A sends a challenge C_A, i.e., a random number, to node B. Node B sends back a message composed by its identifier ID_B, a challenge C_B and the message authentication code computed over a constant k_1, ID_A, ID_B, C_A and C_B. Then a keyed hash function with the master key is executed over the two IDs and the challenges. The results is used as a common secret by both the nodes. From this secret the nodes compute two pairwise keys, one used to sign messages and the other used for encryption.

B. TRANSITORY MASTER KEY

Also this family of protocols uses a global secret in order to protect the pairwise key establishment. However, the global key is deleted after a timeout. The assumption at the base of this approach is that an adversary cannot compromise a node in a short period of time. Therefore, by deleting the global key before a proper timeout the network can be considered safe.

After the deployment, a node has the master key in its memory. This period is defined *initialization phase*. After the deletion of the master key the node starts the *working phase*. Without special techniques, it is possible to establish keys only within the initialization phase, since two nodes need to share the master key. Therefore, adding new nodes after the first deployment would be impossible. However, key management schemes can use specific techniques to provide the possibility to add new nodes able to establish pairwise keys with the previously deployed ones.

If an adversary compromises a node and steals the master key from its memory, he/she can decrypt all the messages exchanged for the pairwise key establishment. Otherwise, if an adversary compromises a node after the deletion of the master key, he/she cannot get advantages for eavesdropping on the other links in the network.

A critical point is represented by the timeout for the deletion of the master key. If the timeout is too long an adversary could be able to compromise the master key. Otherwise, if the timeout is too short the nodes could not be able to establish all the pairwise keys.

The most important scheme based on a transitory master key is LEAP+ [12]. This protocol allows the generation of different types of keys (e.g. a key to exchange messages with the base station, the pairwise keys, etc.). However, the establishment of pairwise keys represents the base of the scheme while all the other keys are derived from these ones.

In [18], the use of a transitory master secret was mixed with the random distribution of keys. This approach provides a high level of security, since an adversary needs both the

transitory secret and the correct key to eavesdrop on a link. However, the computational and communication overheads are high.

C. PUBLIC-KEY CRYPTOGRAPHY

Some key management schemes use public-key cryptography to protect the pairwise key establishment. Transport Layer Security (TLS) represents the basic approach. This protocol is commonly used on generic computer networks to establish secure communications. In a typical WSN implementation of TLS [19], each node has a couple of public and private keys, a certificate that ensures that they are authentic and the public key of the administrator. To generate a common pairwise key, two nodes exchange their certificates and check their authenticity by using the administrator's public key. After the authenticity check, they generate a pairwise key with a public-key agreement algorithm, like Diffie-Hellman. In the rest of the paper this scheme is referred to as Simplified TLS (STLS).

STLS provides a high level of security, since the adversaries cannot forge a certificate and they cannot take any advantage by compromising a node. However, the public-key operations require a great computational effort. Moreover, each asymmetric operation executed by a normal node would be very slow. Therefore, these approaches are often considered too complex for WSNs.

III. TRANSITORY MASTER KEY TRANSPORT LAYER SECURITY

This section presents *Transitory Master Key Transport Layer Security* (TMKTLS), a key management scheme based on a transitory master key. The main goal of this scheme is to achieve most of the benefits of public key cryptography and transitory master key approaches with the limited resources available in WSNs. Therefore, TMKTLS is designed to require lower computational overheads than basic public key schemes and to provide a better security level than transitory master key schemes.

In the proposed scheme, the transitory master key is used for the authentication of the public credentials of the nodes. Each node demonstrates the authenticity of its identity and its public key with a message authentication code, which is indexed with the transitory master key. After a timeout, each node deletes the transitory master key, so possible adversaries cannot forge the signatures.

In STLS, the verification of digital signatures requires a high computational effort, greater than the one required by key agreement. In TMKTLS, the computation of a message authentication code requires negligible time, considerably reducing the computational time needed to authenticate a node. In transitory master key schemes, if a node is compromised before the deletion timeout, the adversary is potentially able to eavesdrop on all the links and to introduce into the network new nodes that pass any authenticity check. TMKTLS has the same protection against eavesdropping of Diffie-Hellman, since it is used to generate the pairwise keys.

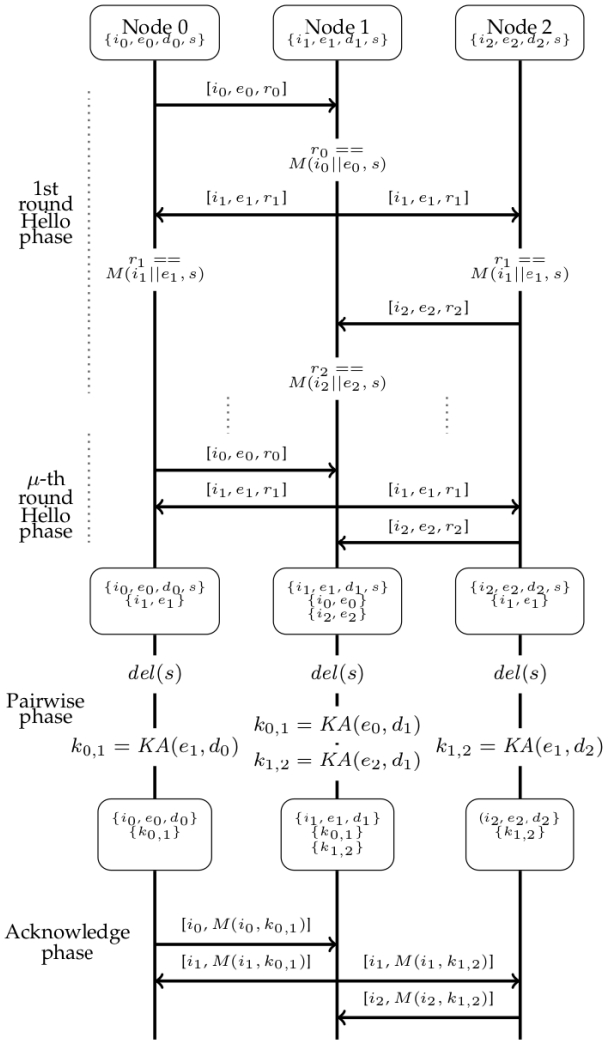

FIGURE 1. Proposed scheme.

The introduction of malicious nodes is possible, since the authentication is based on a global secret, but it can be detected by using the asymmetric cryptography.

The key establishment of TMKTLS is divided into four main phases:

- *Hello Phase*: hello messages are broadcasted by the nodes to perform the neighboring discovery. The nodes check the authenticity of the received messages. At the end of this phase, the transitory master key is deleted;
- *Pairwise Phase*: each node computes pairwise keys with each of the authenticated neighboring nodes;
- *Acknowledge Phase*: acknowledge messages are exchanged in order to confirm each link;
- *Working Phase*: the nodes can still establish new pairwise keys by using public-key digital signatures instead of the transitory master key.

Fig. 1 shows an example of the proposed key establishment. In this case, node 1 can communicate with node 0 and 2. However, node 0 and node 2 cannot communicate directly since they are too far. The data represented between curly

brackets are stored into the memory of the nodes, while the ones in square brackets are sent over the wireless channel.

A. NOTATION AND ASSUMPTIONS

The proposed scheme is based on the following assumptions: there is no deployment knowledge; all nodes are homogeneous; nodes can roam into the network and can be added after the initial deployment; an adversary can eavesdrop on all the messages, inject packets and replay older messages; an adversary can compromise a node and obtain all the data stored in it. After a node is compromised, the adversary has the total control of that node.

The following symbols are used in the rest of the paper:

- n: number of nodes in the network;
- i_x: identifier of a generic node x;
- e_x, d_x: public and private keys of node x;
- s: transitory master key;
- $k_{x,y}$: pairwise key between node x and node y;
- c_x: digital signature in the public certificate of node x;
- e_{CA}: public key of the network administrator;
- r_x: signature of x's hello message;
- $M()$: message authentication code function;
- $DS()$: digital signature;
- $KA()$: key agreement function;
- $||$: concatenation operator.

B. PREDEPLOYMENT PHASE

Before deploying the network, all nodes are preloaded with initial data. The data assigned to node u are i_u, e_u, d_u and s. Each node also knows the functions $M()$ and $KA()$.

At the boot, a node computes its hello message, which is composed of i_u, e_u and a signature corresponding to the *message authentication code* computed over the concatenation of i_u and e_u, indexed by s: $r_u = M(i_u||e_u, s)$. Since the hello message of a node is based on static data, it can be computed just once.

C. FIRST PHASE: NEIGHBOR DISCOVERY

During the first phase, the nodes look for possible neighboring nodes, in order to establish secure links. Each node periodically broadcasts an hello message in order to communicate to the other nodes its credentials.

The neighbor discovery phase is composed by μ rounds, with a duration of t_h per round. The total duration of the first phase is $T_h = \mu \cdot t_h$. Each node broadcasts its hello message once per round. The message is sent at a random instant of each round, in order to decrease possible collisions.

Node v sends an hello message composed of $[i_v, e_v, r_v]$. After receiving this message, node u checks the received signature. If $r_v == MAC(i_v||e_v, s)$, the message is authentic, since v knows s. Node u saves i_v and e_v into its memory.

The duration of this phase should be chosen carefully, depending on the network characteristics. If it is too short, not all the nodes will be able to establish a pairwise key with all their neighboring nodes and the network connectivity decreases. If it is too long, the probability for an adversary to capture a node and compromise the transitory key increases.

In Fig. 1, node 0 broadcasts its hello message and node 1 receives and verifies its authenticity; then, node 1 and 2 repeat the same operation. As it can be seen in the μ-th round, if a certain hello message was already received, it is not verified again.

D. SECOND PHASE: PAIRWISE KEY ESTABLISHMENT

The goal of the second phase is the establishment of the pairwise keys. Then, since the authentication has been completed in the previous phase, the transitory key is deleted at the beginning of this phase. Each node computes a pairwise key per authenticated neighboring node. At the end of the phase each link will have a unique symmetric key, known only by a pair of nodes.

The establishment of the pairwise keys is protected by public key cryptography. TMKTLS is compliant with key agreement functions based on a Diffie-Hellman key exchange (e.g. Elliptic Curve Diffie-Hellman, ECDH). These public key functions allow two devices to generate a common secret, which can be used as symmetric key. For example, in order to establish a pairwise key between nodes u and v, node u computes $KA(d_u, e_v)$ while node v computes $KA(d_v, e_u)$. Both the nodes generate the same common secret $k_{u,v}$.

In many contexts, but in particular for WSNs, Diffie-Hellman-like functions represent a proper solution, since to establish a key it is sufficient to know the other party's public key and to exchange few messages [20].

In Fig. 1, each node deletes s from its memory and then computes the pairwise keys for every discovered neighbors by using $KA()$. In this case node 0 and 2 just compute the key with node 1, while node 1 computes both the pairwise keys.

E. THIRD PHASE: KEY ACKNOWLEDGEMENT

During the third phase, a 2-way acknowledgement is executed in order to confirm the correct generation of the pairwise keys. The goal of this phase is to ensure that the nodes can correctly communicate with each other. The acknowledgement also allows to detect malicious nodes that participated to the previous phases without the correct credentials. In every pair of nodes, the one with the lowest identification number is defined *initiator* while the other one is defined *recipient*. As an example, let's consider node u as the initiator and node v as the recipient. The initiator starts the acknowledgment: it signs its identification number i_u with $M(i_u, k_{u,v})$ and sends this message to the recipient. The recipient answers with i_v signed with $M(i_v, k_{u,v})$. If both u and v correctly verify the received signatures it means that the pairwise key $k_{u,v}$ was correctly created.

In the example, node 0 initiates the acknowledgment with node 1, that responds accordingly; node 1 does the same with node 2. The message authentication code verification is not represented in Fig. 1.

F. FOURTH PHASE: KEY ESTABLISHMENT WITHOUT THE TRANSITORY KEY

When a node is added to the network after the initial deployment or it is moved, it is unable to establish pairwise keys with the nodes that have already deleted s. Therefore, a key establishment among nodes in the working phase is required. This routine is based on public certificates.

Each node uses its own digital signature c_u and the network administrator's public key e_{CA}, that is used to verify other certificates.

First of all, each node in the working phase periodically broadcasts its *discovery* message, in the same way as in the first phase of the protocol. This node becomes the initiator of the key establishment. Assuming that node u is the initiator, the discovery message is composed of $[i_u, e_u, c_u]$, where c_u is equal to the digital signature $DS(i_u||e_u, d_{CA})$, computed over the message that has to be sent with the private key of the certificate authority, that in this case is the network administrator. If a node that does not share a pairwise key with the initiator receives a discovery message, it immediately checks the received message. Then, if it is authentic, the node responds with its own certificate and starts to compute the pairwise key. The answer is sent in unicast, to reduce the computation done by the remaining nodes in the network.

Then the initiator verifies all the received discovery messages. For example, it will verify v's message with $V(i_v||e_v, c_v, e_{CA})$, where $V()$ is the complementary function of $DS()$; it takes the message and the signature as input and verifies them against the public key e_{CA}. If the function succeeds, it means that v is a valid node, so the pairwise key is computed with the function $KA()$, as in the second phase.

Finally, each new pairwise key is confirmed with a 2-way acknowledgment, as in the third phase.

G. MALICIOUS NODE DETECTION

If the transitory key is compromised, an adversary can introduce into the network malicious nodes able to establish keys with other nodes with s. This possibility is limited, since each node will recognize as authentic the malicious nodes only for the short time before deleting s. However, a malicious node detection routine can identify them.

The general idea is to execute a key establishment without the transitory key, since only an authentic node can generate a pairwise key in that way. In order to verify the authenticity of a suspect node, an inquirer node sends a *check* message. This message has the same content of the discovery message used in the fourth phase. The suspect node checks the received message and answers with a discovery message. The inquirer node checks the received message. If the message is valid, both the nodes compute a shared key with the function $KA()$. Finally, a 2-way acknowledgment is performed in order to verify the authenticity of the suspect node.

IV. EVALUATION AND COMPARISON

In this section, the proposed algorithm is analyzed from the theoretical point of view and compared with other state-of-the-art protocols. The analysis is focused on security, memory and computation.

TABLE 1. Resilience comparison. All the schemes provide the maximum of the minimum level.

Phase	Threat	FPWK	LEAP+	TMKTLS
Initialization phase	Forgery	MAX	MIN	MIN
	Eavesdropping	MAX	MIN	MAX
Working phase	Forgery	MAX	MAX	MAX
	Eavesdropping	MAX	MAX	MAX

A. CONNECTIVITY

The connectivity level is here defined as the probability of successfully establishing a link with a neighboring node. The maximum level of connectivity is reached if every node is able to communicate with all its neighboring nodes. The minimum, if no node is able to communicate.

LEAP+ can provide the maximum connectivity only if the neighbor discovery phase is long enough, since a routine for the establishment of pairwise keys between two nodes without the transitory key is not present.

FPWK, STLS and TMKTLS guarantee that all links are created, so the level of connectivity is the maximum. In particular, in FPWK each node knows the key for each link. In STLS the asymmetric cryptography allows all the nodes to establish a pairwise key with any other node. In TMKTLS, although the nodes could be unable to complete the key establishment within a too short timeout, they can always establish the pairwise keys without the transitory key.

B. RESILIENCE

The resilience level is defined as the ability of resisting to compromised secret material. In particular, the resilience against *eavesdropping* is computed according to the probability that an adversary cannot eavesdrop on a link, while the resilience against node *forgery* is identified by the probability that an adversary cannot pass an authentication check. The possibility to eavesdrop on the links of the compromised nodes or to clone the compromised nodes are not considered. The maximum level of resilience against eavesdropping is reached if no link can be eavesdropped. The minimum if all the links can be eavesdropped. The maximum level of resilience against forgery is reached if all nodes can recognize the fake nodes. The minimum if all the nodes cannot recognize them. Table 1 shows the level of resilience against eavesdropping (*eavesdropping* rows) and against node forgery (*forgery* rows) if an adversary has compromised nodes within the initialization or the working phases.

FPWK and STLS always provide the maximum level of resilience. This level is reached since a node does not store any information useful to find the keys generated by the other nodes.

In LEAP+, if at least a node is compromised during the initialization phase all the links could be eavesdropped and

TABLE 2. Recoverability comparison. All the schemes provide the maximum of the minimum level.

Phase	FPWK	LEAP+	TMKTLS
Initialization phase	MAX	MIN	MAX
Working phase	MAX	MAX	MAX

the adversary can forge new nodes able to pass all the authenticity checks. If some nodes are compromised in the working phase, the level of resilience is maximum.

In TMKTLS, the transitory key is the only secret that can be stolen from a node in order to attack other parts of the network. Therefore, TMKTLS provides the maximum resilience during the working phase. During the initialization, the level of resilience against eavesdropping is maximum, since all the pairwise keys are protected by asymmetric cryptography. However, an adversary with the transitory master key can introduce malicious nodes able to establish a pairwise key with nodes that still store the transitory key.

C. RECOVERABILITY

The recoverability level is defined as the ability of restoring secure communication after some compromised secret material has been revoked [21]. It can be computed according to the probability that a link is still safe or a new key can be established to protect it. The maximum level of recoverability is reached if every node will be able to communicate with all its neighboring nodes after revoking the compromised secret material. The minimum if no node can communicate after revoking the secret material. Table 2 shows the level of recoverability if an adversary has compromised nodes within the initialization or the working phases.

FPWK, STLS and LEAP+ have a level of recoverability equal to the level of resilience, which is always the maximum or the minimum. Therefore, FPWK and STLS always provide the maximum level of recoverabilty, while LEAP+ provides the maximum if a node is compromised during the working phase, and the minimum if a node is compromised during the initialization phase.

If a node is compromised during the working phase, TMKTLS provides the maximum level of recoverability, as for the resilience. If a node is compromised during the Initialization phase, the adversary is able to introduce new malicious nodes. Nevertheless, the malicious node detection routine allows to check the authenticity of the nodes and to revoke the keys used to communicate with them. Therefore, the maximum level of recoverability is provided.

D. COMPUTATIONAL REQUIREMENTS

The operations involved by the analyzed schemes are shown in Table 3, where: v represents the verification of a public key signature, dh represents a shared secret computation using the Diffie-Hellman scheme, r represents a keyed pseudo-random function, and m represents a message authentication code or a hash function.

TABLE 3. Computational requirements comparison.

Protocol	Required operations
FPWK	negligible
STLS	$v + dh + 2m$
LEAP+	$r + m + r$
TMKTLS	$m + dh + 2m$

TABLE 4. Memory requirements comparison.

Protocol	Memory
FPWK	$(n-1)l_k$
STLS	$l_d + 2l_e + l_s + v(l_k + l_{id})$
LEAP+	$l_k + v(l_k + l_{id})$
TMKTLS	$l_d + 2l_e + l_s + v(l_k + l_{id})$

The most efficient scheme is FPWK, since all the pairwise keys are preloaded and no operation is required after the deployment. LEAP+ requires to execute two pseudo-random functions and a message authentication code in order to compute the other node individual master key, the pairwise key and to verify a message. The involved computational overheads are still low.

In STLS, the operations involved are the verification of a digital signature and two message authentication codes. TMKTLS requires a Diffie-Hellman scheme execution and three message authentication codes. Both in STLS and TMKTLS, two of the message authentication codes are needed for the acknowledgement task. Since the computational overheads of a message authentication code execution are extremely lower than public key operations, STLS requires more computational time than TMKTLS, which requires more time than LEAP+.

E. MEMORY REQUIREMENTS

Table 4 shows the memory required by each scheme in order to store the secret material. The number of nodes in the network and the number of nodes in direct communication are n and v, respectively. The length of secret material are: l_k for the symmetric keys, l_d for the private keys, l_e for the public keys, l_s for the public digital signatures, l_m for the message authentication codes, and l_a for the temporary key. The length of the identification number of a node is l_{id}.

FPWK has the largest memory overhead, since each node stores a key per node of the network, independently from the density. During the working phase, in all the other schemes, each node stores the pairwise keys with its neighboring nodes, for an area that corresponds to $v(l_k + l_{id})$. Among these schemes, LEAP+ has the lowest memory overhead, since each node stores only two keys during the initial phase. STLS has larger requirements, since each node stores a pair of asymmetric keys for the communication, the digital signature and the public key of the certificate authority. TMKTLS has larger memory requirements than STLS, since each node stores the same material as in STLS plus a message authentication code signature and the transitory master key.

However, during the working phase, the additional secret material is deleted.

F. OVERALL COMPARISON

The analyzed schemes are all based on similar assumptions. They all allow, to some extent, to add nodes after the initial deployment (only FPWK has some limitations in this area because it requires unused pairwise keys); they also provide a high level of security when nodes are compromised during the working phase of the protocols.

In order to provide a quantitative comparison, the following case study is considered (all values are in bytes): $l_k = 16$, $l_d = 20$, $l_e = 40$, $l_s = 40$, $l_m = 4$, $l_a = 48$ and $l_{id} = 1$,; the limit to the memory size dedicated to the secret material is 4 KB.

According to the formulas in Table 4, FPWK is compliant with networks composed by at most 256 nodes, STLS and TMKTLS with networks with at most 232 nodes in direct communication and LEAP+ 240 nodes in direct communication. Therefore, LEAP+, STLS and TMKTLS are compliant with large networks with high density.

From the computational point of view, FPWK is the scheme with the lowest requirements. LEAP+ has larger requirements, but they are still faster than the protocols based on public key cryptography. TMKTLS has lower requirements than STLS, which is the scheme with the largest requirements.

Although the schemes based on random predistribution cannot reach the maximum resilience, if some nodes are compromised during the working phase all the considered schemes provide a high level of security. In particular in FPWK, LEAP+, STLS and TMKTLS, an adversary that compromises a node doesn't obtain useful information about other nodes, so he/she cannot eavesdrop on any link or introduce new nodes different from the compromised ones. If some nodes are compromised during the initialization phase, in LEAP+ the adversary could eavesdrop on all the links and introduce new nodes. In TMKTLS, an adversary cannot eavesdrop on any link. However, he/she could introduce new nodes able to establish a link with nodes that still store the transitory key. Nevertheless, these fake nodes will be able to operate only until a malicious node detection.

For medium to large size WSNs with high density, TMKTLS represents a suitable key management scheme, since it provides a high level of security with computational effort lower than STLS.

V. EXPERIMENTAL ANALYSIS

A prototypical implementation of TMKTLS has been developed on the TinyOS platform in order to evaluate its feasibility on the current generation of nodes and verify its performances. The main goal of this analysis is to compare TMKTLS and STLS, in order to correctly evaluate the execution times of the proposed protocol; LEAP+ was not considered because, from the performance point-of-view, it will always be faster that protocols based on public-key cryptography. Furthermore, resilience was not considered in this analysis because it is a static property that does not affect the experimental comparison.

The prototypical implementation was executed on the TelosB Tmote Sky IV, a sensor node that mounts a MSP430 8 MHz microcontroller with 10 KB RAM and a CC2420 wireless chip with a transmission rate of 250 kbps. From the software point-of-view, TMKTLS was developed with the NesC language, a C dialect specifically designed for TinyOS; the library TinyECC was used to provide support for elliptic curve cryptography, in particular for the Elliptic Curve Diffie-Hellman (ECDH) algorithm. Moreover, Crypto library was used to for the Multilinear Modular Hashing (MMH), a fast message authentication code algorithm.

The actual implementation of TMKTLS has some small differences with the described algorithm that, however, do not affect the overall analysis. In particular, public and private key pairs are generated on-line at the node boot-up instead of being preloaded into the node memory.

An implementation of STLS was also developed in order to perform the comparison with TMKTLS; these implementations are very similar: STLS uses a public key digital signature instead of a message authentication code, that is verified during the pairwise phase for each of the neighboring nodes.

The focus of this analysis is on the optimization of the neighbor discovery, in order to reach a high connectivity and on the execution time of the algorithm, to verify the benefits with respect to STLS.

A. HELLO PHASE OPTIMIZATION

Optimizing the hello phase means choosing the minimal duration that guarantees the desired average level of connectivity. Here, connectivity is the ratio between successfully created links and the total number of possible links among nodes. Shortening the initial phase also implies a higher security of the protocol.

The neighbor discovery phase, as explained in section III-C, is divided into rounds, in which nodes broadcast their hello messages at random instants. Using probability methods it is possible to analyze this procedure and compute the average connectivity:

$$C_{avg} = \left(1 - \left(1 - \left(1 - 2\frac{t_{tx}}{t_h}\right)^{v-1}\right)^{\mu}\right)^2 \quad (1)$$

where the involved parameters are: t_{tx} as the duration of the transmission, t_h as the round duration, v as the number of neighboring nodes and μ as the number of rounds. In order to correctly receive a whole message, even partial overlap with other messages must be avoided. Therefore, in order to receive the hello message from a node within a round, the fraction of time in which no other message can be sent is $2\frac{t_{tx}}{t_h}$. Considering that each node sends an hello message per round, the probability that in a round the hello message from a node has no collision is $\left(1 - 2\frac{t_{tx}}{t_h}\right)^{v-1}$. The probability that the hello message from a node has at least a collision in every

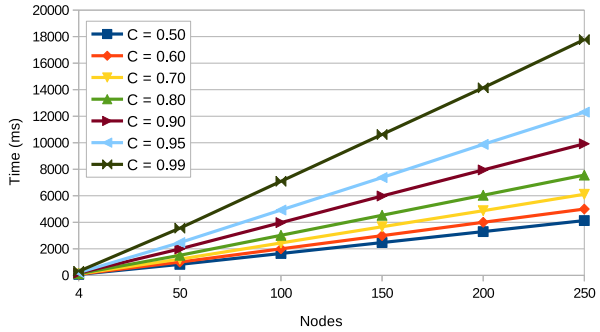

FIGURE 2. Neighbor discovery minimum duration of TMKTLS according to the target connectivity.

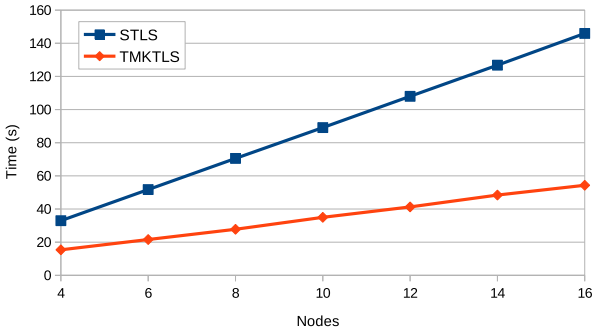

FIGURE 3. Total execution time.

TABLE 5. TMKTLS neighbor discovery final settings according to the number of neighboring nodes.

Nodes	Period (ms)	Rounds	Total (ms)
4	50	5	250
6	50	7	350
8	80	6	480
10	80	7	560
12	120	6	720
14	80	10	800
16	80	12	960

round is $\left(1-\left(1-2\frac{t_{tx}}{t_h}\right)^{v-1}\right)^{\mu}$. The final square operation is required since both the involved nodes have to successfully receive the reciprocal message. In order to verify the proposed formula, a in-house simulator was developed in Python programming language. The tests were run 30 times with random parameters, each case was simulated 10^6 times and, on average, the difference with the formula was below 1%.

Using (1), it is possible to obtain, through a simple lookup table, the best values of t_h and μ for a given t_{tx}, v and the desired C_{avg}.

Figure 2 shows the minimal neighbor discovery duration given the number of neighboring nodes, for different target connectivity levels; the transmission time was set to 3.2 ms, that corresponds to the time required by a TelosB sensor node to send 100 bytes. Even for more than 250 nodes, the minimum neighbor discovery duration is about 18 s; if the target connectivity is lowered to 0.95, the duration becomes about 12 s. This shows that, even for high density networks, the initial phase is short, so the time window in which the protocol is vulnerable is small. The number of neighboring nodes v depends on the network density; for instance, if the density is 1 (i.e. each node is able to directly communicate with every other node), v is equal to the number of nodes in the network.

These optimizations were used for the analysis of the execution time of TMKTLS, in order to minimize the discovery phase. A target connectivity of 0.99 was set, and t_{tx} was set to 2.33 ms, that is the transmission time of the hello message (73 B) on the TelosB sensor node. Table 5 shows the values used for the TMKTLS discovery phase, for different number of nodes. They were obtained with the method explained in this section; the actual round duration was rounded up to an integer number for implementation reasons.

B. EXECUTION TIME ANALYSIS

Figure 3 shows the total execution time of TMKTLS and STLS (neighbor discovery, pairwise and acknowledge phase), with a network with a number of nodes from 4 to 16, adding 2 nodes each time; the target connectivity was set to the maximum, so that each node is able to communicate with every other node in the network. The values in this section are the average execution time over 5 experiments for each number of nodes.

The neighbor discovery phase of TMKTLS was optimized with the values reported in table 5, while STLS uses the same approach but with different values, that are due to a longer hello message.

The pairwise phase may have a variable duration on different nodes because of an intrinsic variability of the ECDH algorithm (on TelosB motes, 2971 $\pm$ 33.9 ms). This requires that the acknowledgment phase is long enough to allow all nodes to finish at different instants but still have enough time to perform all the acknowledgments. Because of this, the acknowledgment phase duration was set to a value that is proportional to the ECDH variability and the number of nodes plus a fixed time; in general it goes from 5 to 8 seconds.

As we can see, TMKTLS execution time is drastically lower than STLS, for any number of nodes. The ratio between the two slopes is 0.35; this means that TMKTLS has an execution time that is about 35% of STLS. This result is mainly due to the pairwise phase duration, which is longer in STLS; for each neighbor node, in TMKTLS one ECDH computation is done, while in STLS one digital signature verification and one ECDH are executed. Considering that the average ECDH execution is of 2971 ms, while verifying a certificate takes 5889 ms, if we compute 2971 / (5889 + 2971), the result equal to 0.34 confirms the previous analysis. The duration is extremely linear with respect to the number of nodes for both TMKTLS and STLS; this is because all the three phases of the algorithms are dependent on the network density.

VI. CONCLUSION

In this paper, a new key management scheme for WSNs, TMKTLS, has been presented. It is based on a hybrid approach, in which a transitory master key is used to authenticate nodes in the network, while pairwise keys are created using public cryptography. The main benefits of TMKTLS are the lower computational overheads with respect to TLS-based approaches, and a higher level of resilience with respect to transitory master key schemes.

A comparative theoretical and experimental analysis showed that TMKTLS provides good security properties and that its computational overheads are compliant with real WSNs. Therefore, TMKTLS can be considered a valuable solution, especially for WSNs with strict security constraints.

As a future work, the proposed approach will be texted on a larger network.

REFERENCES

[1] M. T. Lazarescu and L. Lavagno, "Wireless sensor networks," in *Handbook of Hardware/Software Codesign*. Berlin, Germany: Springer, 2017, pp. 1–42.
[2] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, "Internet of Things security: A survey," *J. Netw. Comput. Appl.*, vol. 88, pp. 10–28, Jun. 2017.
[3] Y. Luo, Y. Duan, W. Li, P. Pace, and G. Fortino, "A novel mobile and hierarchical data transmission architecture for smart factories," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3534–3546, Aug. 2018.
[4] W. Lu, Y. Gong, X. Liu, J. Wu, and H. Peng, "Collaborative energy and information transfer in green wireless sensor networks for smart cities," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1585–1593, Apr. 2018.
[5] A. Velasco, R. Ferrero, F. Gandino, B. Montrucchio, and M. Rebaudengo, "A Mobile and Low-Cost system for environmental monitoring: A case study," *Sensors*, vol. 16, no. 5, p. 710, May 2016.
[6] M. Collotta, G. Scata, S. Tirrito, R. Ferrero, and M. Rebaudengo, "A parallel fuzzy scheme to improve power consumption management in Wireless Sensor Networks," in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2014, pp. 1–4.
[7] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Distributed channel allocation protocols for wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2264–2274, Sep. 2014.
[8] Z. Ruan, H. Luo, and Z. Chen, "Improving reliability of erasure codes-based storage paradigm under correlated failures for wireless sensor networks," *Int. J. Commun. Syst.*, vol. 29, no. 5, pp. 992–1011, 2016.
[9] Y. Zhang, Y. Shen, H. Wang, J. Yong, and X. Jiang, "On secure wireless communications for IoT under eavesdropper collusion," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 3, pp. 1281–1293, Jul. 2016.
[10] A. K. Mishra and A. K. Turuk, "A comparative analysis of node replica detection schemes in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 61, pp. 21–32, Feb. 2016.
[11] C.-Y. Chen and H.-C. Chao, "A survey of key distribution in wireless sensor networks," *Secur. Commun. Netw.*, vol. 7, no. 12, pp. 2495–2508, Dec. 2014.
[12] S. Zhu, S. Setia, and S. Jajodia, "LEAP+ Efficient security mechanisms for large-scale distributed sensor networks," *ACM Trans. Sensor Netw.*, vol. 2, no. 4, pp. 500–528, 2006.
[13] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, document 5246, Internet Engineering Task Force, 2008.
[14] M. Griotti, F. Gandino, and M. Rebaudengo, "Mixed public and secret-key cryptography for wireless sensor networks," in *Proc. 10th Int. Conf. Mobile Comput. Ubiquitous Netw. (ICMU)*, Oct. 2017, pp. 1–6.
[15] O. Cheikhrouhou, "Secure group communication in wireless sensor networks: A survey," *J. Netw. Comput. Appl.*, vol. 61, pp. 115–132, Feb. 2016.
[16] X. He, M. Niedermeier, and H. De Meer, "Dynamic key management in wireless sensor networks: A survey," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 611–622, Mar. 2013.
[17] *ZigBee Specification Document 053474r20*, ZigBee Alliance Standard, 2012.
[18] F. Gandino, B. Montrucchio, and M. Rebaudengo, "Key management for static wireless sensor networks with node adding," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1133–1143, May 2014.
[19] S. K. Sahoo and M. N. Sahoo, *An Elliptic-Curve-Based Hierarchical Cluster Key Management in Wireless Sensor Network* India, New Delhi: Springer, 2014, pp. 397–408.
[20] H. Wang, B. Sheng, and Q. Li, "Elliptic curve cryptography-based access control in sensor networks," *Int. J. Secur. Netw.*, vol. 1, no. 3/4, p. 127, 2006.
[21] F. Gandino and A. Servetti, "Key recoverability in wireless sensor networks," *IEEE Access*, vol. 7, pp. 164407–164417, 2019.

MATTIA GRIOTTI received the B.S. and M.S. degrees in computer engineering from the Politecnico di Torino, in 2014 and 2016, respectively. He is currently a Software Engineer with TecSA s.r.l., where he is developing real-time embedded C++ software.

FILIPPO GANDINO (Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering from the Politecnico di Torino, in 2005 and 2010, respectively. He is currently an Associate Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include ubiquitous computing, RFID, WSNs, security and privacy, network modeling, and quantum computing.

MAURIZIO REBAUDENGO (Senior Member, IEEE) received the M.S. degree in electronics and the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, in 1991 and 1995, respectively. He is currently a Full Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interest includes ubiquitous computing and testing and dependability analysis of computer-based systems.